Build a Text Classification Model with Attention Mechanism NLP Project Overview

Business Overview

Attention is an increasingly popular mechanism used in many applications today. The attention mechanism focus on dynamically highlighting relevant features of the input data. Here, the model gives more weightage to the most relevant word in a sentence to make predictions.

In the previous projects, we have seen how to implement the Naïve Bayes algorithm, skip-gram model, Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM) models for text classification.

In this project, let's see how well does the attention model work for our text classification problem.

Aim

To perform multiclass text classification on the dataset with the help of the attention mechanism

Data Description

The dataset contains more than two million customer complaints about consumer financial products. Amongst the various available columns, we have a column that contains the actual text of the complaint and one column containing the product for which the customer is raising the complaint.
We have also used pre-trained word vectors - glove.6B
The link to download this data is as follows:
https://nlp.stanford.edu/projects/glove/

Tech Stack

- ➔ Language: Python
- ➔ Libraries: pandas, torch, nltk, numpy, pickle, re, tqdm, sklearn

Prerequisite

1. The torch framework
2. Multiclass Text Classification using Naive Bayes in Python
3. Skip Gram Model Python Implementation for Word Embeddings
4. Build Multi Class Text Classification Models with RNN and LSTM

Approach

1. Installing the necessary packages through the pip command
2. Importing the required libraries
3. Defining configuration file paths
4. Process glove embeddings
    - Read the text file
    - Convert embeddings to float array
    - Add embeddings for padding and unknown items
    - Save embeddings and vocabulary

5. Process Text data
    - Read the CSV file and drop the null values
    - Replace duplicate labels
    - Encode the label column and save the encoder and encoded labels

6. Data Preprocessing
    - Conversion to lower case
    - Punctuation removal
    - Digits removal
    - Remove more than one consecutive instance of 'x'
    - Additional spaces removal
    - Tokenize the text
    - Save the tokens
7. Model Building
    - Create attention model
    - Create a function for the PyTorch dataset
    - Function to train the model
    - Function to test the model

8. Train function
    - Load the files
    - Split data into train, test, and validation
    - Create PyTorch datasets
    - Create data loaders
    - Create model object
    - Move the model to GPU if available
    - Define loss function and optimizer
    - Training the model
    - Test the model

9. Prediction of new text

Modular Code Overview

```
Input
  |_complaints.csv
  |_glove.6B.50d.txt


Output
  |_attention.pth
  |_embeddings.pkl
  |_label_encoder.pkl
  |_labels.pkl
  |_vocabulary.pkl
  |_tokens.pkl

Source
  |_data.py
  |_model.py
  |_utils.py


|_config.py
|_Engine.py
|_predict.py
|_processing.py
|_README.MD
|_requirements.txt
|_attention.ipynb
```

Once you unzip the modular_code.zip file, you can find the following folders within it.

1. Input
2. Output
3. Source

1. The input folder contains the data that we have for analysis. In our case, it
    - complaints.csv
    - glove.6B.50d.txt (download it from the link provided)

2. The source folder contains all the modularized code for all the above steps in a modularized manner. It includes the following.
    - model.py
    - data.py
    - utils.py

    These all-python files contain helpful functions which are being used in the Engine.py file.

3. The output folder contains the following;
    - attention.pth
    - embeddings.pkl
    - label_encoder.pkl
    - labels.pkl
    - vocabulary.pkl
    - tokens.pkl

    These are required for our model training and can be quickly loaded and used for future use, and the user need not have to train all the models from the beginning.

4. The config.py file contains all the configurations required for this project.

5. The Engine.py file is the main file that needs to be called to run the entire code in one go. It trains the model and saves it in the output folder.
   Note: Please check the README.md file for more information.

6. The attention.ipynb is the original notebook we saw in the videos.

7. The processing.py file is used to process the data.

8. The predict.py file is used to make predictions on the data.

9. The README.md file contains all the information on how to run particular files and more instructions to follow.

10. The requirements.txt file has all the required libraries with respective versions. Kindly install the file by using the command **pip install -r requirements.txt**

Project Takeaways

1. Understanding the business problem
2. Introduction to attention mechanism
3. Understanding the working of attention mechanism
4. What are pre-trained word vectors?
5. Steps to process glove embeddings
6. Data preparation for the models
7. How do remove spaces and digits?
8. Removing the punctuations
9. How to create the architecture for the attention model?
10. How to create a data loader for the attention model?
11. Building the attention model
12. Training attention model using CUDA or CPU
13. How to make predictions on the new text data?
14. Learn the concept of how to use attention on RNN